Instructions for Creating an EtherNet/IP connection to SICK Auto ID devices

Change History

Version	*Editor*	*Changes / Comments*	*Date*
0.1	W. Lee	Draft	2017-04-17
1.0	W. Lee	Release	2017-08-25
1.1	W. Lee	Corrected foot notes for the Input Data Map	2018-02-06
1.2	W. Lee	Modified with Explicit Message	2019-04-19
1.3	W. Lee	Updated the data map for the Input and Output	2019-11-12
1.4	W. Lee	Modified Output Data Format (Removed LECTOR Dynamic Focus)	2021-08-02

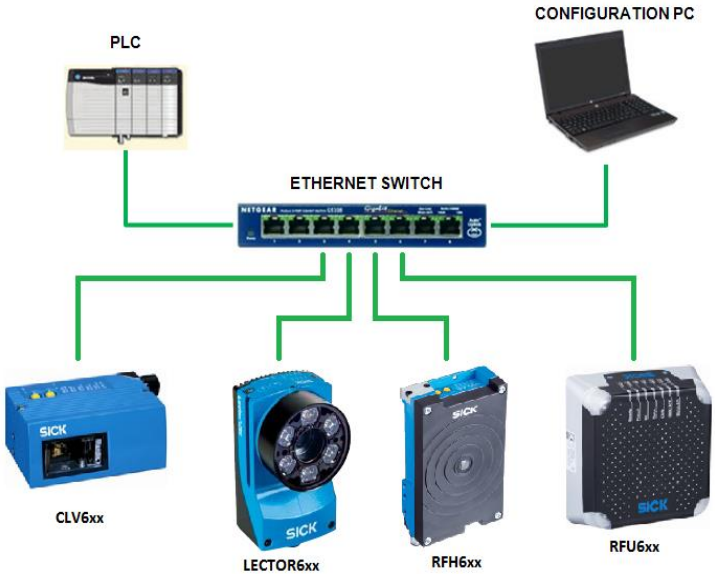
CONTENT

I. Creating an EtherNet/IP connection

This is a procedure to establish a connection between SICK Auto ID Devices to a programmable controller over EtherNet/IP. The following procedure can be used with a SICK CLV6xx LASER Bar code reader, LECTOR6xx Image bar code reader, RFU6xx UHF RFID Interrogator, and RFH6xx HF RFID Interrogator.

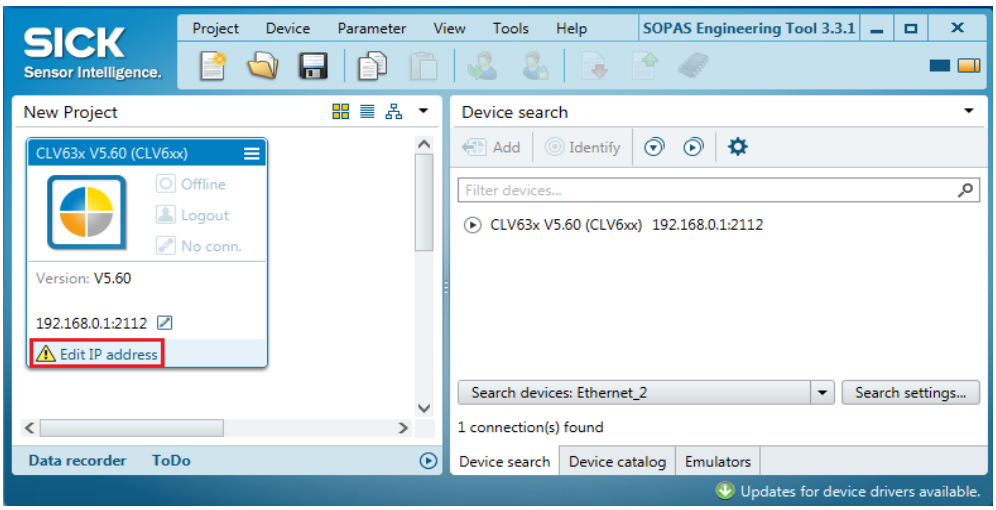
All SICK Auto ID devices use the same connection type based on a Request Packet Interval (RPI) on EtherNet/IP. SOPAS parameterization is identical, and data generated for all platforms are similar.

SICK Auto ID devices are used in the star topology as shown.

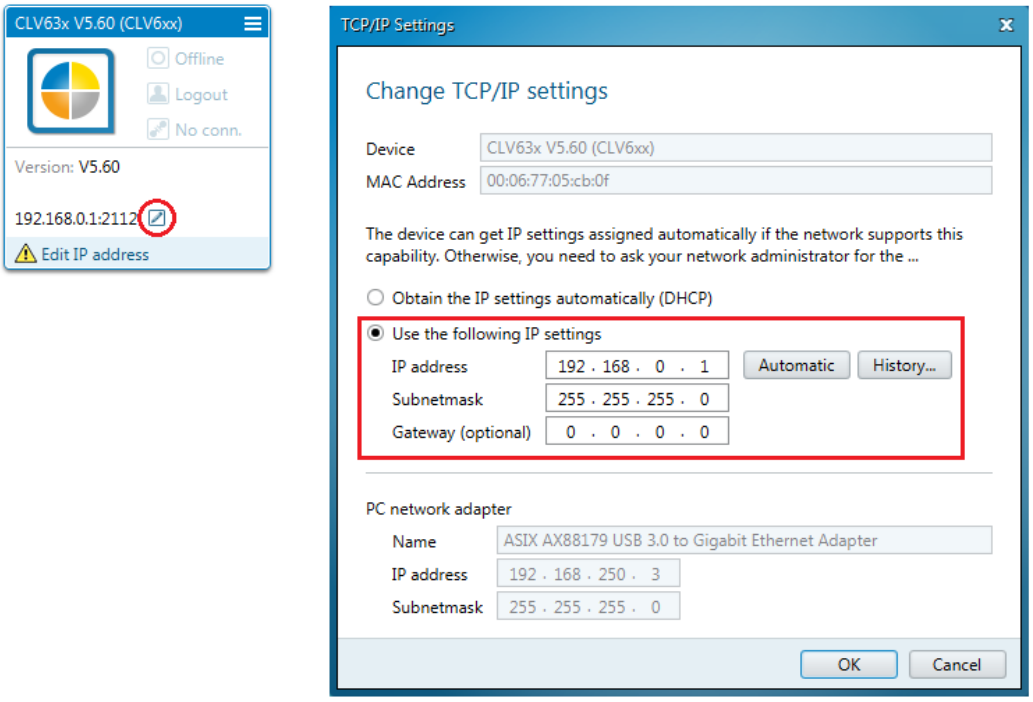


As mentioned the parameterization of SICK Auto ID devices are the same. The software used to configure SICK Auto ID devices is SOPAS-ET (SICK Open Portal Application Systems – Engineering Tool).

Initially the device should be assigned an IP Address so that it is on the same subnet as the programmable controller.

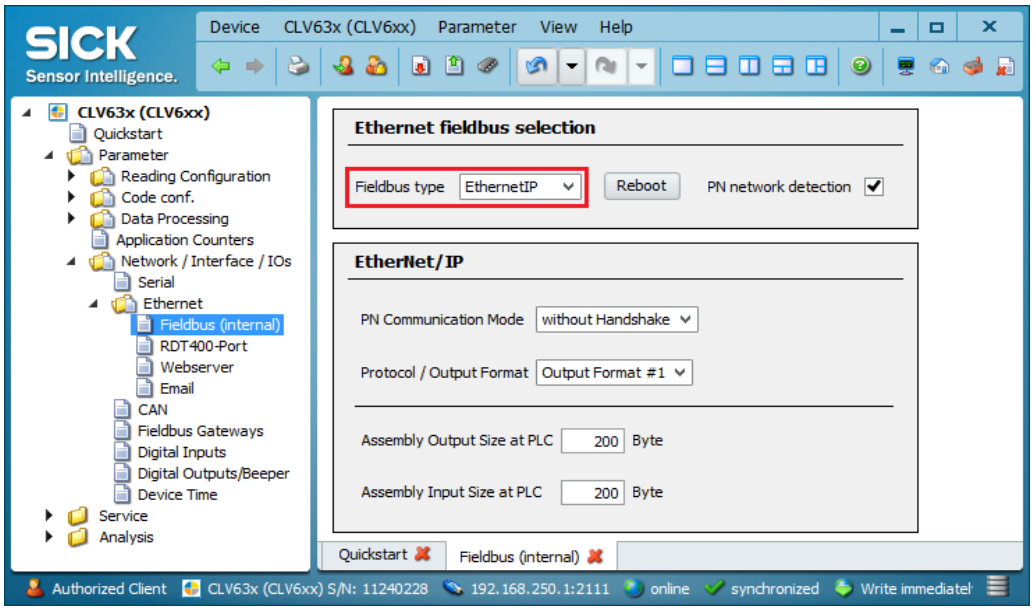


The SICK Auto ID device should be assigned a static IP Address.



However, SICK Auto ID devices are capable of receiving an IP Address from a DHCP server.

After an IP Address has been assigned to the device EtherNet/IP must be selected in the parameter ‘Fieldbus (internal)’. This will activate EtherNet/IP capability within the device.



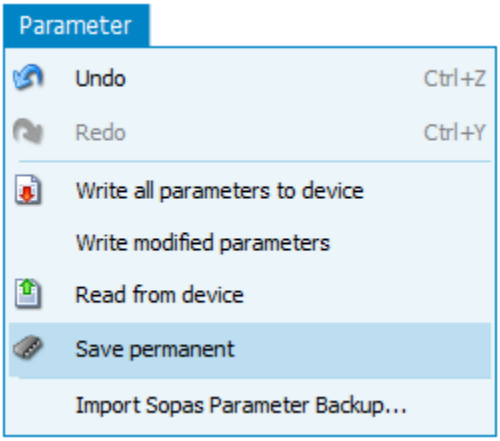
Once EtherNet/IP has been selected for the fieldbus type its properties will become available. The values shown in the EtherNet/IP properties are the default values.

SICK Auto ID devices supports two communication modes. These modes describe how the strings (read results and SOPAS command language commands/responses) are exchanged between the programmable controller and the device over EtherNet/IP

- With Handshake
- Without Handshake

NOTE: The handshaking mechanism (see Appendix) is used to improve security. As soon as multiple data telegrams (e.g. multiple scanners on CAN multiplexed with one scanner over EtherNet/IP) may occur at the same time, the handshaking is required to avoid data loss.

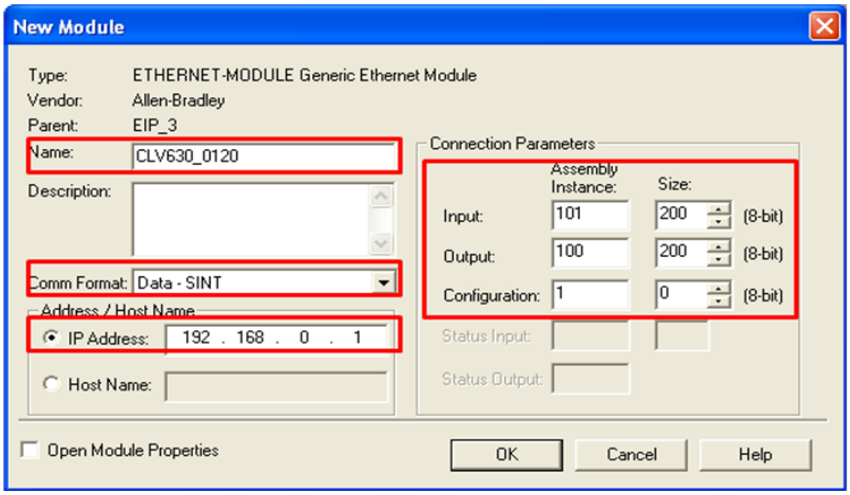
These changes should be saved to the device.



II. Input and Output Data Assemblies

SICK Auto ID devices connect to the programmable controller by way of a CIP (Common Industrial Protocol) connection. There are two ways to setup SICK Auto ID devices in the I/O Configuration of the PLC.

The first is to use the '**Generic Ethernet Module'**.



THE INFROMATION PROVIDED HERE IS THE NAME OF THE DEVICE, DATA FORMAT, IP ADDRESS, AND DATA SIZE.

The information required to configure the CIP connection in the programmable controller are.

Name	"Device Name"
Comm Format	Data – SINT (Single-INTeger)
IP Address	192.168.0.1 (Device Default)

	Assembly Instance	**Size**
Input	101	200
Output	100	200
Configuration	1	0

Where:

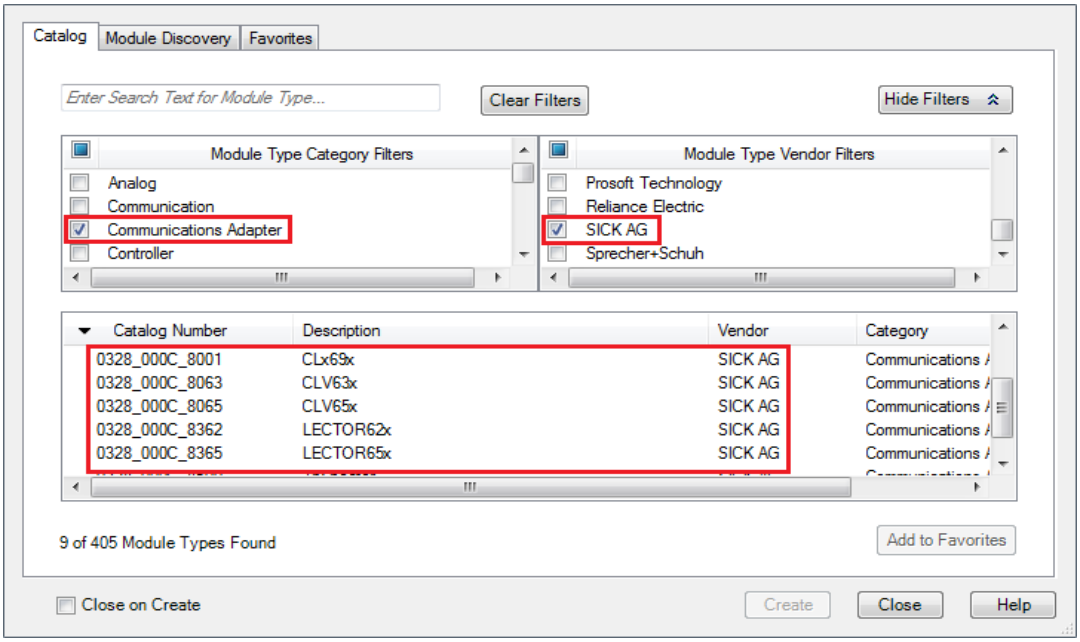
Input is data from the SICK device to the PLC.

Output is data from the programmable controller to the SICK device.

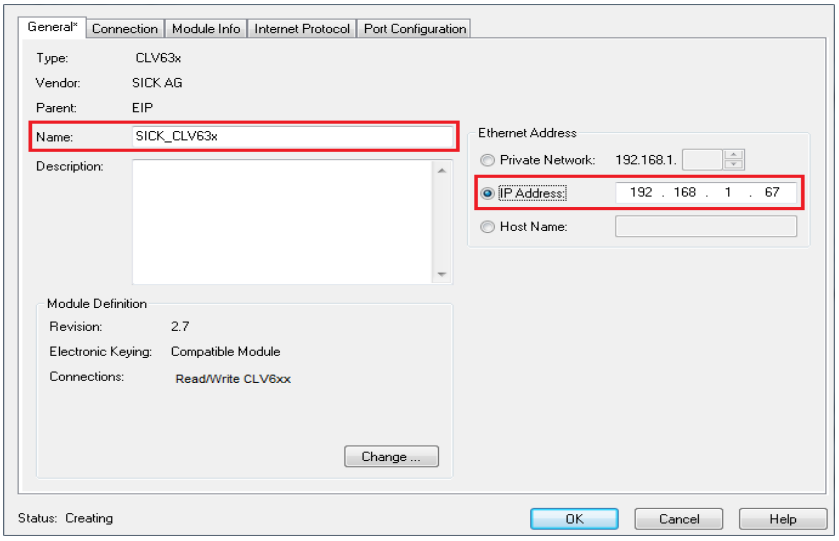
The data exchanged between the SICK device and the programmable controller includes one Input Assembly Instance and one Output Assembly Instance both with a fixed size of 200 Bytes (the Configuration Assembly is just a dummy and not actually used, therefore enter the values as shown in the table above).

The second method is to use the Electronic Data Sheet (**EDS**) file. This file provides the programmable controller information about the SICK device as well as the amount of data the device produces and consumes.

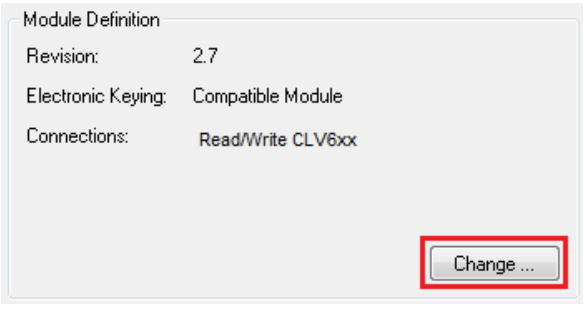
To be able to configure the SICK device its **EDS** file must be registered first into the programmable controller. Once the **EDS** file has been registered the device will then become available in the programmable controller's configuration software.

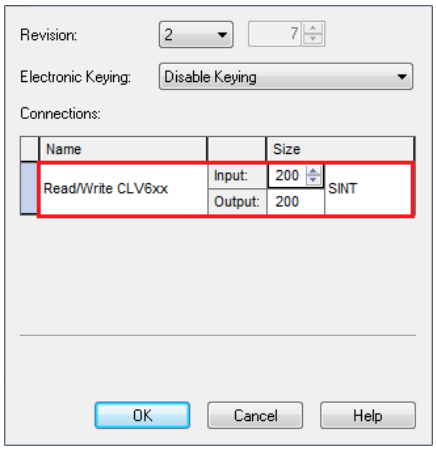


When the SICK device is selected from the programmable controller's configuration software "**Catalog**" as a '**Communication Adapter**' with '**SICK AG**' being the vendor, its properties window will open so that its configuration can be completed.



Where access to modify the Input and Output data from the SICK device. Under the '**Module Definition**' window select the "**Change…**" button.



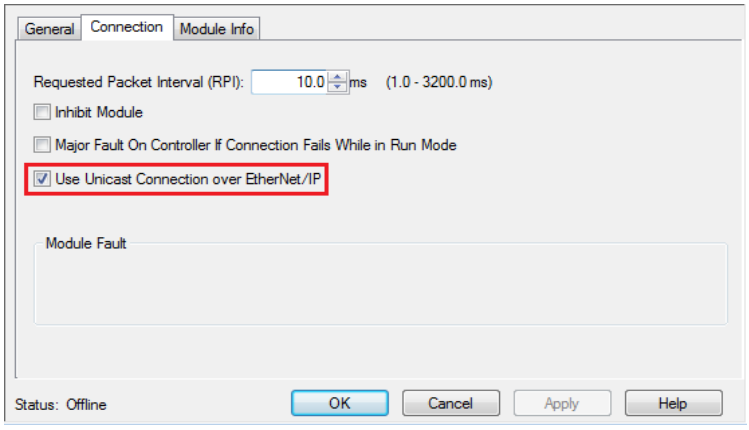


The primary difference between the **'Generic Ethernet Module'** configuration and the **EDS** file configuration are the data types generated.

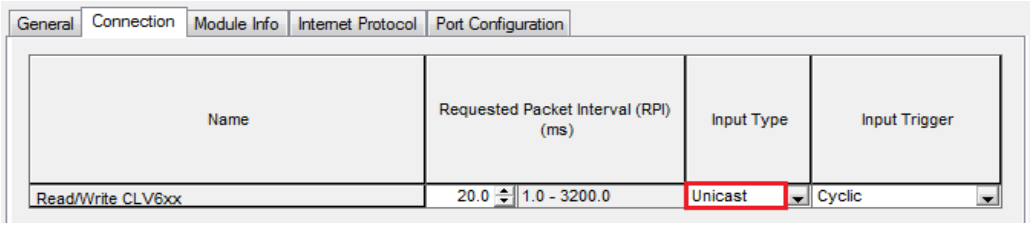
The '**Generic Ethernet Module**' configuration will generate one Input Tag Database, one Output Tag Database, and one Configuration Tag Database within the programmable controller. The **EDS** file configuration will generate one Input Tag Database and one Output Tag Database. These databases can be found in the programmable controller's "**Controller Tags**".

Whether using the '**Generic Ethernet Module**' or the EDS Profile the connection must be setup to use '**Unicast**'.

'**Generic Ethernet Module**' properties



'**EDS Profile**' properties



A. Input Data Format

The Input Data Format is described in the following table. This table shows the Byte allocation as well as the Bit description for the entire Input Data Format for SICK Auto ID devices.

Status Word	Fragmentation Protocol Header				InTelegram
	Input Status	InTelegram Count	OutTelegram CountBack	InTelegram LenRest	
16 bits	8 bits	8 bits	8 bits	16 bits	1544 bits (193 Bytes)

DATA TYPE	WORD	BIT	DESCRIPTION	COMMENTS			
				CLV6xx	LECTOR6xx	RFH6xx	RFU6xx
Status Word	BYTE 0	D0	Device Ready	Yes	Yes	Yes	Yes
		D1	System Ready	Not yet implemented			
		D2	Good Read	Yes	Yes	Yes	Yes
		D3	No Read	Yes	Yes	Yes	Yes
		D4	Status Ext Output 1	Yes[1]	Yes	Yes	Yes
		D5	Status Ext Output 2	Yes[1]	Yes	Yes	Yes
		D6	Status Output 1 (Result 1)	Yes	Yes	Yes	Yes
		D7	Status Output 2 (Result 2)	Yes	Yes	Yes	Yes
	BYTE 1	D0	Ext Input 1	Yes[1]	Yes	Yes[1]	Yes[1]
		D1	Ext Input 2	Yes[1]	Yes	Yes[1]	Yes[1]
		D2	Input 1 (Sensor 1)	Yes	Yes	Yes	Yes
		D3	Input 2 (Sensor 2)	N/A	Yes	Yes	Yes
		D4	Status Output 3 (Result 3)	Yes[2]	N/A	N/A	N/A
		D5	Status Output 4 (Result 4)	Yes[2]	N/A	N/A	N/A
		D6	Reserved	-	-	-	-
		D7	GRNR Toggle**	Yes	Yes	Yes	Yes
Input Status	BYTE 2	D0	Reserved	-			
		D1	Reserved	-			
		D2	Heartbeat	The heartbeat bit shows the devices' presence and availability. The heartbeat will toggle between High (1) and Low (0) in a fixed interval.			
		D3	PLC Error	The SICK device has detected an error in the data communications between itself and the programmable controller. The SICK device denies the data transfer and requests a new synchronization between itself and the controller. The programmable controller may contain errors that lead to the wrong handling. Not used without handshake.			
		D4	Reserved	-			
		D5	Reserved	-			
		D6	BufferOverrun	All output buffers in the SICK device are in use. The device will deny all data transfer			

				with this error. The programmable controller may retry the transmission later and publish the error to a user interface.
		D7	Reserved	-
InTelegram Count	BYTE 3	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
OutTelegram CountBack	BYTE 4	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
InTelegram LenRest (Motorola Format)	BYTE 5	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
	BYTE 6	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
InTelegram	BYTE 7 TO BYTE 199 (193 BYTES)	1544 Bits	User Data	

The yellow fields are available and the blue fields are reserved for future use.
[1]Only on the CLV63x-65x, RFH and RFU only available with External parameter storage module (CMC600).

[2]Outputs only available on the CLV69x.

****Toggles for each reading gate.**

Device Ready – The current state of the SICK AutoID device is set to 0 while booting up or while configuring. Set to 1 when the device is ready. “Ready” indicates that the internal microprocessor is ready. It is not meant to be an indicator of all possible hardware faults. It will turn on, along with the “Ready LED”.

Good Read – Active as soon as data is read. This will be reset when no data is read during a subsequent reading gate. The criteria for a “Good Read” are configured in SOPAS.

No Read – Active if no data was read during a reading gate. This will be reset with the next “good” read. Not available for continuous reading configuration.

Status External Output 1 and 2 – Represents the current states of the respective outputs.

Status Output 1 (Result 1) and Output 2 (Result 2) – Represents the current states of the respective outputs.

External Input 1 & 2 and Input 1 (Sensor 1) & Input 2 (Sensor 2) – Represents the current states of the respective inputs.

Heartbeat - The heartbeat bit is provided to the fieldbus master to show its presence and availability on the connection. The heartbeat will toggle between High (1) and Low (0) in a fixed interval.

PLC Error - The SICK AutoID device has detected an error in the data communications between device and programmable controller. The device denies the data transfer and requests a new synchronization between itself and the programmable controller. The programmable controller may contain errors that lead to the incorrect handling of data communications. **Not used without handshake**.

Buffer overrun - All output buffers in the SICK AutoID device are in use. The device will deny all data transfer with this error. The programmable controller may retry the transmission later and publish the error to a user interface.

GRNR_Toggle - ‘Good Read No Read’ bit toggles from 0 to 1 every time the reading gate closes.

B. Output Data Format

The Output Data Format is described in the following table. This table shows the Byte allocation as well as the Bit description for the entire Output Data Format for SICK AutoID devices.

Command Word	Fragmentation Protocol Header				OutTelegram
	Output Status	InTelegram CountBack	OutTelegram Count	OutTelegram LenRest	
16 bits	8 bits	8 bits	8 bits	16 bits	1544 bits (193 Bytes)

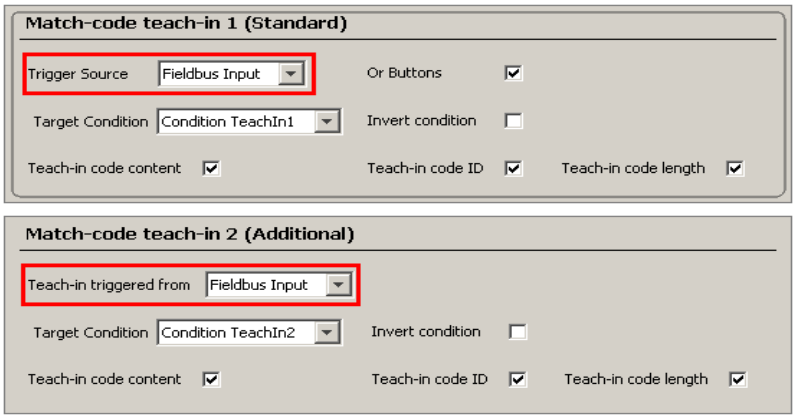
DATA TYPE	WORD	BIT	DESCRIPTION	COMMENTS			
				CLV6xx	LECTOR6xx	RFH6xx	RFU6xx
Command Word	BYTE 0	D0	Trigger	Yes	Yes	Yes	Yes
		D1	Sensor-Idle	Not yet implemented			
		D2	TeachIn 1	Yes	Yes	N/A	N/A
		D3	TeachIn 2	Yes	Yes	N/A	N/A
		D4	Ext Output 1	Yes[1]	Yes[1]	Yes[1]	Yes[1]
		D5	Ext Output 2	Yes[1]	Yes[1]	Yes[1]	Yes[1]
		D6	Digital Output 1 (Result 1)	Not yet imple-mented	Yes	Not yet imple-mented	Yes
		D7	Digital Output 2 (Result 2)	Not yet imple-mented	Yes	Not yet imple-mented	Yes
	BYTE 1	D0	PLC_Out_08	Not yet implemented			
		D1	PLC_Out_09	Not yet implemented			
		D2	PLC_Out_10	Not yet implemented			
		D3	PLC_Out_11	Not yet implemented			
		D4	Distance Config_0 (LSB)	Yes[2]	N/A	N/A	N/A
		D5	Distance Config_1	Yes[2]	N/A	N/A	N/A
		D6	Distance Config_2	Yes[2]	N/A	N/A	N/A
		D7	Distance Config_3 (MSB)	Yes[2]	N/A	N/A	N/A
Output Status	BYTE 2	D0	Reserved	-			
		D1	Reserved	-			
		D2	Reserved	-			
		D3	Reserved	-			
		D4	Reserved	-			
		D5	Reserved	-			
		D6	Reserved	-			
		D7	Reserved	-			
InTelegram CountBack	BYTE 3	D0					
		D1					
		D2					
		D3					
		D4					
		D5					
		D6					
		D7					

OutTelegram Count	BYTE 4	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
OutTelegram LenRest (Motorola Fromat)	BYTE 5	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
	BYTE 6	D0		
		D1		
		D2		
		D3		
		D4		
		D5		
		D6		
		D7		
OutTelegram	BYTE 7 TO BYTE 199 (193 BYTES)	1544 Bits	Commands	

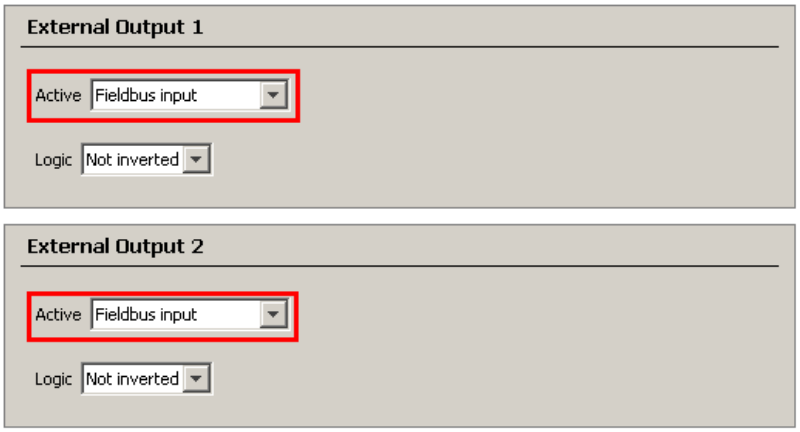
The yellow fields are available and the blue fields are reserved for future use.
[1]On the CLV63x-65x, RFH and RFU only available with External parameter storage module (CMC600).
[2]Only available on the CLV64x, CLV65x, and CLV69x.

Trigger – Triggers the reading gate if the device is set to trigger from 'Fieldbus Input ' (see Appendix for more info).

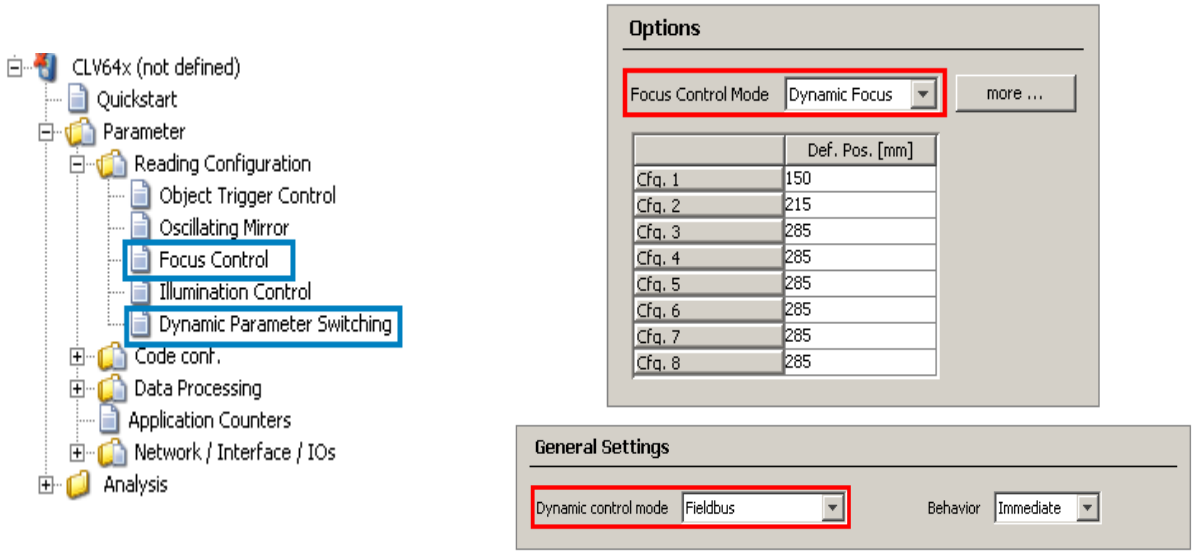
TeachIn 1 & 2 – These bits trigger a teach-in according to the configuration on the Match-code teach-in within SOPAS. To do this set the Teach-In Trigger Source to 'Fieldbus Input'



External Output 1 & 2 – Controls the outputs of the device when configured to 'Fieldbus Inputs'.

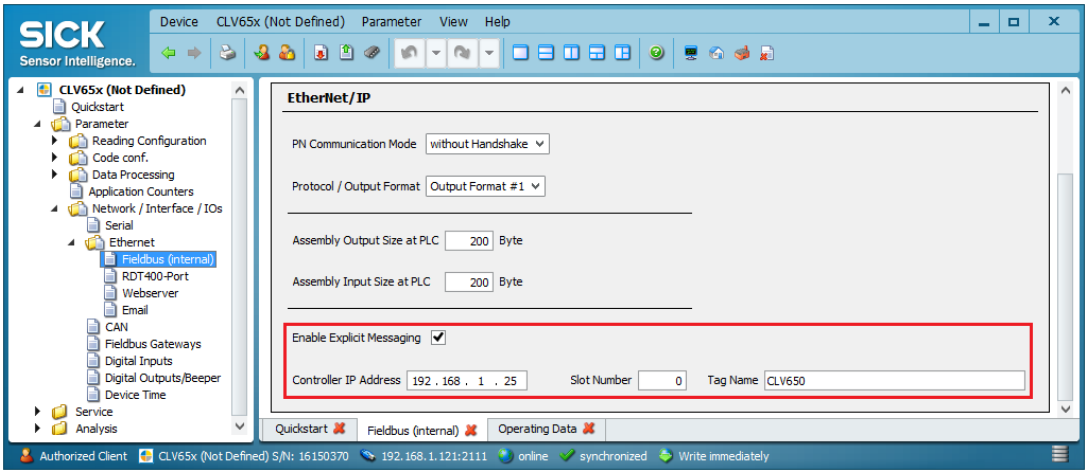


Distance_Config_0 thru 3 – For devices with dynamic focus, these bits may control the focus position if enabled in SOPAS on the Focus Control page. This is applicable to the CLV64x, CLV65x, and CLV69x. To do this set 'Focus Control Mode' to 'Dynamic Focus' and select 'Dynamic Control mode' to 'Fieldbus'.

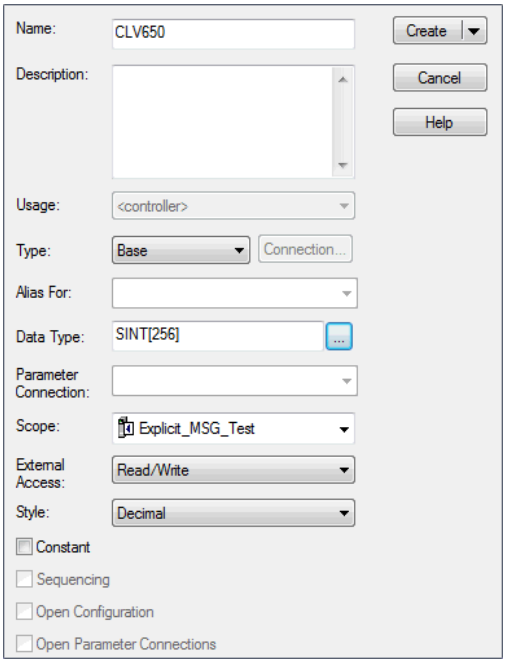


III. Explicit Message Setup

The CLV62x - 65x has Explicit Massaging capability. This is enabled as shown.



Once this is enabled a User Defined Tag will need to be created in the programmable controller where the bar code data from the CLV6xx will populate.



The data would be populated into the UDT like this.

WORD	Description
Byte 0	Data count
Byte 1	Data size
Byte 2 - 255	CLV62x – 65x Data

IV. APPENDIX

A. Handshaking Example

The input data, in this example coming from the SICK AutoID device to the field bus master (Programmable Controller) is "**CLV 630**". It has a length of nine characters including the special 'Start' and 'End' characters. The counters are equal, and shows that any prior data transmission have been completed in both directions.

NOTE: Configure EtherNet/IP in SOPAS to use Communication Mode "**with Handshake**".

1. **Starting situation: No New Data Available**
2. **Incoming data from the SICK AutoID device to the programmable controller**
3. **Programmable controller responds to the data from the device**

TAG NAME	STYLE	DATA TYPE	VALUES		
			STP 1	STP 2	STP 3
Input Telegram					
SICK_EIP:I.Data[0]	Decimal	Status Word	69	69	69
SICK_EIP:I.Data[1]	Decimal	Status Word	0	0	0
SICK_EIP:I.Data[2]	Decimal	Input Status	4	4	4
SICK_EIP:I.Data[3]	Decimal	InTelegramCount	0	1	1
SICK_EIP:I.Data[4]	Decimal	OutTelegramCountBack	0	0	0
SICK_EIP:I.Data[5]	Decimal	InTelegramLenRest (MSB)	0	0	0
SICK_EIP:I.Data[6]	Decimal	InTelegramLenRest (LSB)	0	9	9
SICK_EIP:I.Data[7]	ASCII	InTelegram (User Data)	'$00'	'$02'	'$02'
SICK_EIP:I.Data[8]	ASCII	InTelegram (User Data)	'$00'	'C'	'C'
SICK_EIP:I.Data[9]	ASCII	InTelegram (User Data)	'$00'	'L'	'L'
SICK_EIP:I.Data[10]	ASCII	InTelegram (User Data)	'$00'	'V'	'V'
SICK_EIP:I.Data[11]	ASCII	InTelegram (User Data)	'$00'	' '	' '
SICK_EIP:I.Data[12]	ASCII	InTelegram (User Data)	'$00'	'6'	'6'
SICK_EIP:I.Data[13]	ASCII	InTelegram (User Data)	'$00'	'3'	'3'
SICK_EIP:I.Data[14]	ASCII	InTelegram (User Data)	'$00'	'0'	'0'
SICK_EIP:I.Data[15]	ASCII	InTelegram (User Data)	'$00'	'$03'	'$03'
SICK_EIP:I.Data[16]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'
SICK_EIP:I.Data[17]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'
SICK_EIP:I.Data[18]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'
SICK_EIP:I.Data[19]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'
Output Telegram					
SICK_EIP:O.Data[0]	Decimal	Command Word	0	0	0
SICK_EIP:O.Data[1]	Decimal	Command Word	0	0	0
SICK_EIP:O.Data[2]	Decimal	Output Status	0	0	0
SICK_EIP:O.Data[3]	Decimal	InTelegramCountBack	0	0	1
SICK_EIP:O.Data[4]	Decimal	OutTelegramCount	0	0	0
SICK_EIP:O.Data[5]	Decimal	OutTelegramLenRest (MSB)	0	0	0
SICK_EIP:O.Data[6]	Decimal	OutTelegramLenRest (LSB)	0	0	0

4. **Example of the field bus master (programmable controller) sending a telegram (Command) to the SICK AutoID device.**

NOTE: This is not an actual SOPAS command it is just ASCII characters for demonstration purposes.

Load the length of the outgoing telegram into Byte 6

At the same time (or immediately following) Write (Copy) the data into the output but beginning at Byte 7 in the **OutTelegram**.

Increment the **OutTelegramCount**.

5. **Response from SICK AutoID device to the programmable controller sending data.**

Transmission between the programmable controller and SICK AutoID device are complete and the counters are equal.

6. **Incoming data from the SICK AutoID device to the programmable controller**

NOTE: Clear the OutTelegram after receiving correct response from SICK AutoID device (e.g.: OutTelegramCountBack incremented).

7. PLC Response post any PLC Processing of incoming data

TAG NAME	STYLE	DATA TYPE	VALUES			
			STP 4	STP 5	STP 6	STP 7
Input Telegram						
SICK_EIP:I.Data[0]	Decimal	Status Word	69	69	69	69
SICK_EIP:I.Data[1]	Decimal	Status Word	0	0	0	0
SICK_EIP:I.Data[2]	Decimal	Input Status	4	4	4	4
SICK_EIP:I.Data[3]	Decimal	InTelegramCount	1	1	2	2
SICK_EIP:I.Data[4]	Decimal	OutTelegramCountBack	0	1	0	0
SICK_EIP:I.Data[5]	Decimal	InTelegramLenRest (MSB)	0	0	0	0
SICK_EIP:I.Data[6]	Decimal	InTelegramLenRest (LSB)	9	9	11	11
SICK_EIP:I.Data[7]	ASCII	InTelegram (User Data)	'$02'	'$02'	'$02'	'$02'
SICK_EIP:I.Data[8]	ASCII	InTelegram (User Data)	'C'	'C'	'A'	'A'
SICK_EIP:I.Data[9]	ASCII	InTelegram (User Data)	'L'	'L'	'B'	'B'
SICK_EIP:I.Data[10]	ASCII	InTelegram (User Data)	'V'	'V'	'C'	'C'
SICK_EIP:I.Data[11]	ASCII	InTelegram (User Data)	' '	' '	'D'	'D'
SICK_EIP:I.Data[12]	ASCII	InTelegram (User Data)	'6'	'6'	'E'	'E'
SICK_EIP:I.Data[13]	ASCII	InTelegram (User Data)	'3'	'3'	'F'	'F'
SICK_EIP:I.Data[14]	ASCII	InTelegram (User Data)	'0'	'0'	'G'	'G'
SICK_EIP:I.Data[15]	ASCII	InTelegram (User Data)	'$03'	'$03'	'H'	'H'
SICK_EIP:I.Data[16]	ASCII	InTelegram (User Data)	'$00'	'$00'	'I'	'I'
SICK_EIP:I.Data[17]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$03'	'$03'
SICK_EIP:I.Data[18]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'	'$00'
SICK_EIP:I.Data[19]	ASCII	InTelegram (User Data)	'$00'	'$00'	'$00'	'$00'
Output Telegram						
SICK_EIP:O.Data[0]	Decimal	Command Word	0	0	0	0
SICK_EIP:O.Data[1]	Decimal	Command Word	0	0	0	0
SICK_EIP:O.Data[2]	Decimal	Output Status	0	0	0	0
SICK_EIP:O.Data[3]	Decimal	InTelegramCountBack	1	1	1	2
SICK_EIP:O.Data[4]	Decimal	OutTelegramCount	1	1	1	1
SICK_EIP:O.Data[5]	Decimal	OutTelegramLenRest (MSB)	0	0	0	0
SICK_EIP:O.Data[6]	Decimal	OutTelegramLenRest (LSB)	10	10	0	0
SICK_EIP:O.Data[7]	ASCII	OutTelegram (User Data)	'$02'	'$02'	'$00'	'$00'
SICK_EIP:O.Data[8]	ASCII	OutTelegram (User Data)	'N'	'N'	'$00'	'$00'
SICK_EIP:O.Data[9]	ASCII	OutTelegram (User Data)	'e'	'e'	'$00'	'$00'
SICK_EIP:O.Data[10]	ASCII	OutTelegram (User Data)	'w'	'w'	'$00'	'$00'
SICK_EIP:O.Data[11]	ASCII	OutTelegram (User Data)	'D'	'D'	'$00'	'$00'
SICK_EIP:O.Data[12]	ASCII	OutTelegram (User Data)	'a'	'a'	'$00'	'$00'
SICK_EIP:O.Data[13]	ASCII	OutTelegram (User Data)	't'	't'	'$00'	'$00'
SICK_EIP:O.Data[14]	ASCII	OutTelegram (User Data)	'a'	'a'	'$00'	'$00'
SICK_EIP:O.Data[15]	ASCII	OutTelegram (User Data)	'?'	'?'	'$00'	'$00'
SICK_EIP:O.Data[16]	ASCII	OutTelegram (User Data)	'$03'	'$03'	'$00'	'$00'
SICK_EIP:O.Data[17]	ASCII	OutTelegram (User Data)	'$00'	'$00'	'$00'	'$00'
SICK_EIP:O.Data[18]	ASCII	OutTelegram (User Data)	'$00'	'$00'	'$00'	'$00'
SICK_EIP:O.Data[19]	ASCII	OutTelegram (User Data)	'$00'	'$00'	'$00'	'$00'

NOTE: If data is expected to exceed a data length of 193 Bytes. Please contact SICK Inc. Application Engineer for information of fragmentation protocol.

B. Inside the Programmable Controller

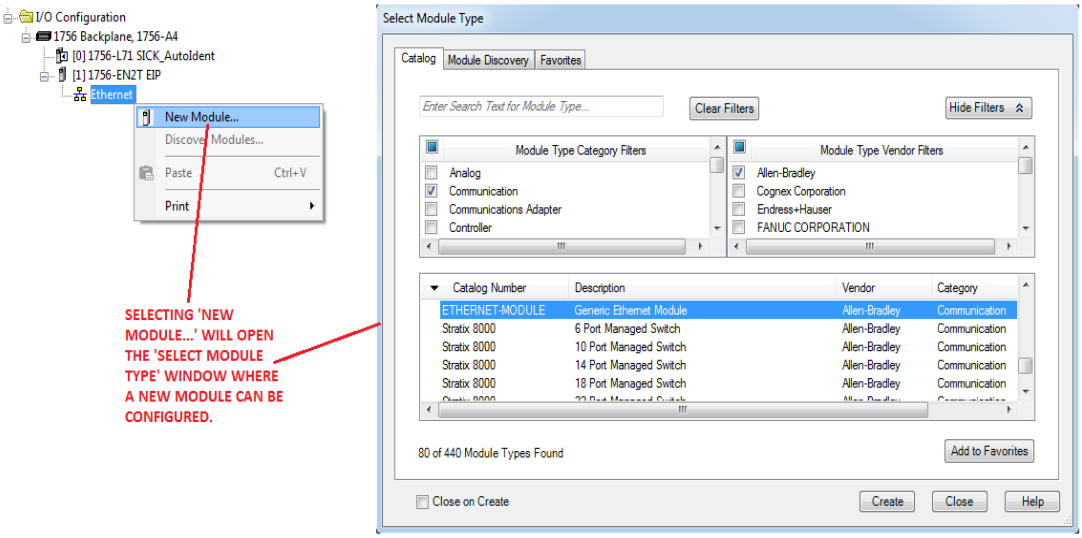
In the following example the CLV630-0120 Bar Code Scanner will be setup in the Rockwell Automation/Allen Bradley Programmable Controller as a '**Generic Ethernet Module**'.

The setup requirements for a NEW configuration (starting with hardware right out of the box) would be to use Rockwell programming software to start a NEW project, configure all the hardware used in the controller (i.e.: local I/O cards and communication modules), and save this configuration to the memory of the programmable controller.

Where the setup of the CLV630-0120 as a '**Generic Ethernet Module**' will look like:

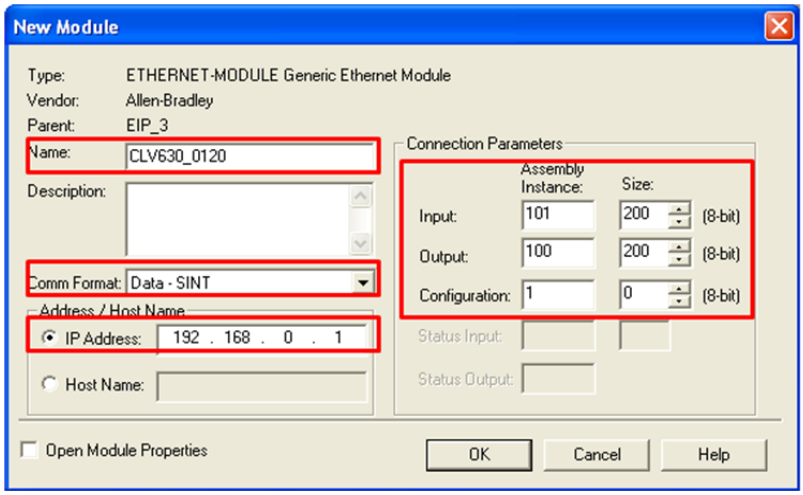
Select '**Ethernet**' under the communication interface 1756-EN2T in the Rockwell programming software and choose '**New Module**' to open the module selection window.

Narrow the search using "Communication" and "Allen Bradley"



Scroll to the '**ETHERNET MODULE**' - '**Generic Ethernet Module**' and select it.

This will open up a '**New Module**' window where the CLV630-0120 data can be entered.

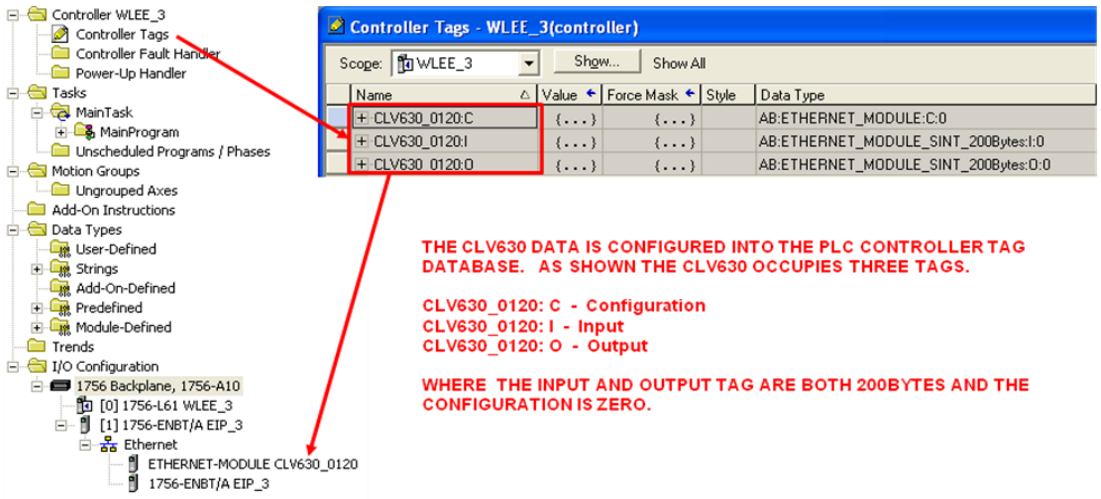


For this example the information required to setup the CLV630-0120 Bar Code Scanner in the programmable controller is as follow. Other than the name this would be the same for all SICK AutoID devices.

Name	CLV630_0120
Comm Format	Data – SINT (Single-INTeger)
IP Address	192.168.0.1 (CLV630 Default)

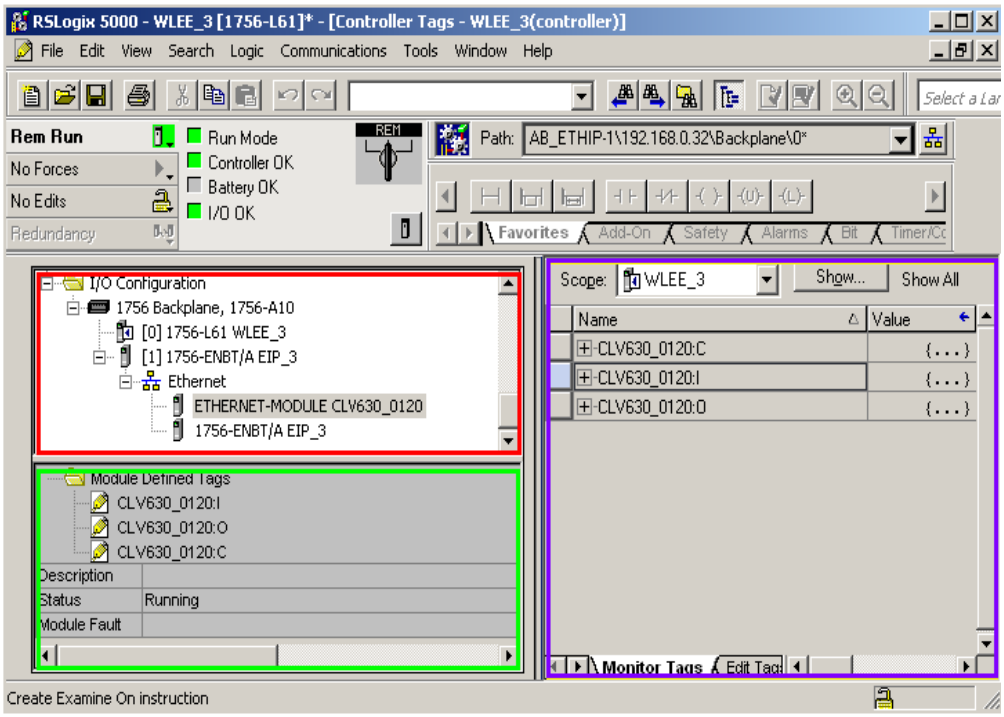
	Assembly Instance	**Size**
Input	101	200
Output	100	200
Configuration	1	0

After the CLV630-0120 is configured, its data will appear as shown.



Rockwell programming software is defaulted into three windows.

- **Controller Organizer**
- **Tag/Main Routine**
- **Diagnostic**



In the '**Controller Organizer**' window under '**I/O Configuration'** if a device is selected its current status will be displayed in the '**Diagnostic**' window. The names of the Tag data generated from the device is shown (if applicable), the Description of the device, the device Status, and Module Fault(s).

Where:

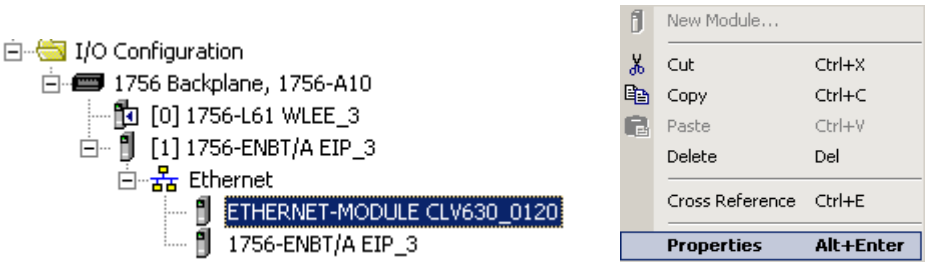
Names of the Tags – are Input, Output, and Configuration data the device generates

Description – user description of the device (this is entered during module setup)

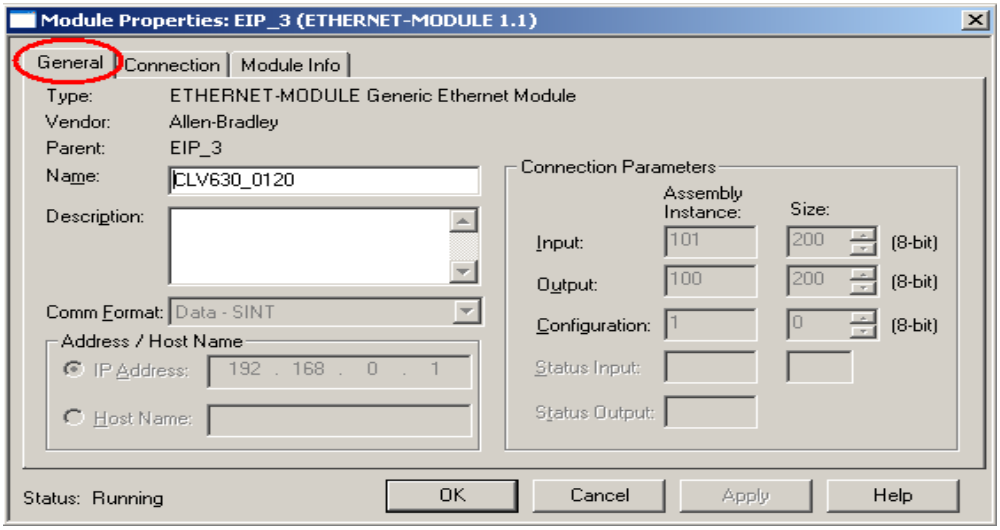
Status – the current condition of the module. Could be 'Running', 'I/O Fault', 'Connecting', 'Shutting Down', etc…

Module Fault – shows the fault code as well as a description of what the controller is attempting to do to correct the problem.

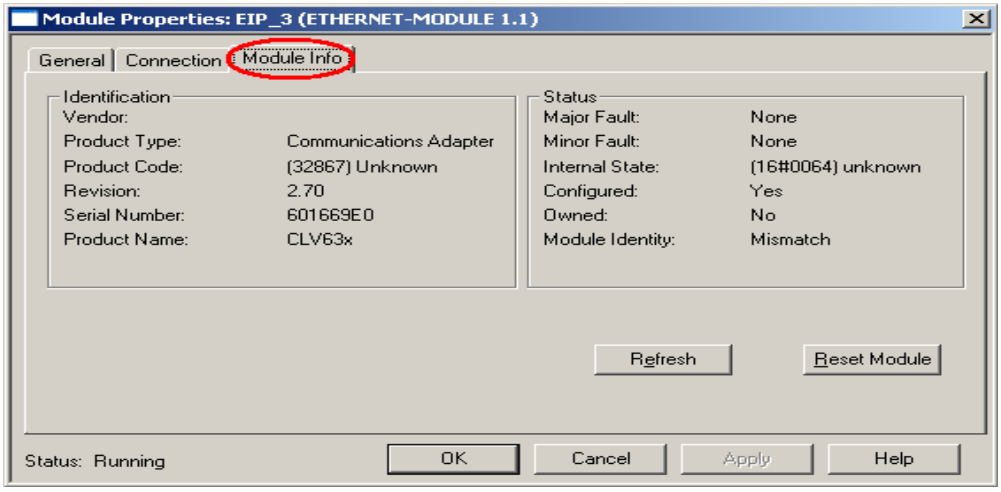
Also, in the '**I/O Configuration**' if a device is selected and 'Right Clicked' a menu will provide access to the device's properties.



In the properties window for the selected device '**General**' module configuration is available

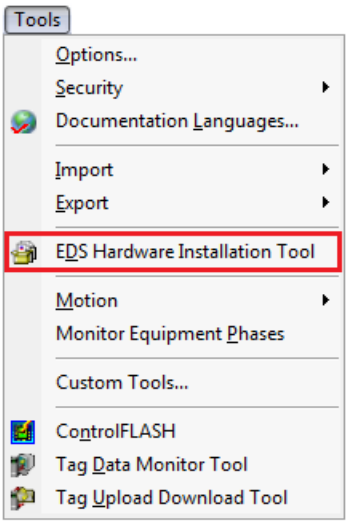


…as well as '**Module Info**'.



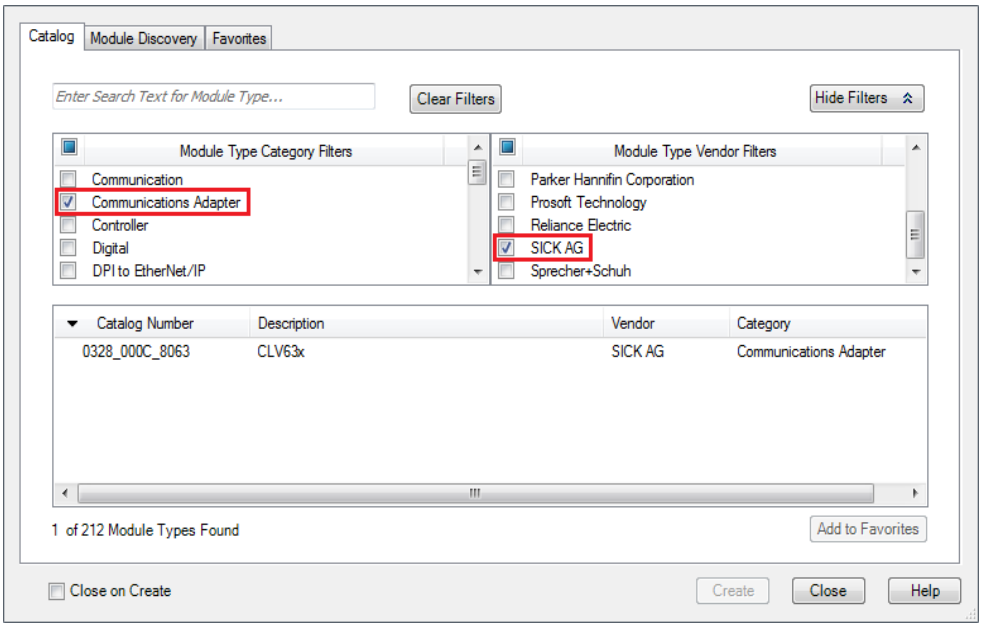
This window provides the current status of the module showing Faults, Hardware Revision, Serial Number, and Product Name.

The alternative configuration would use the Electronic Data Sheet (**EDS**) File. Using the programmable controller's programming software the **EDS** file must first be registered. This can be done using the "**EDS Hardware Installation Tool**".

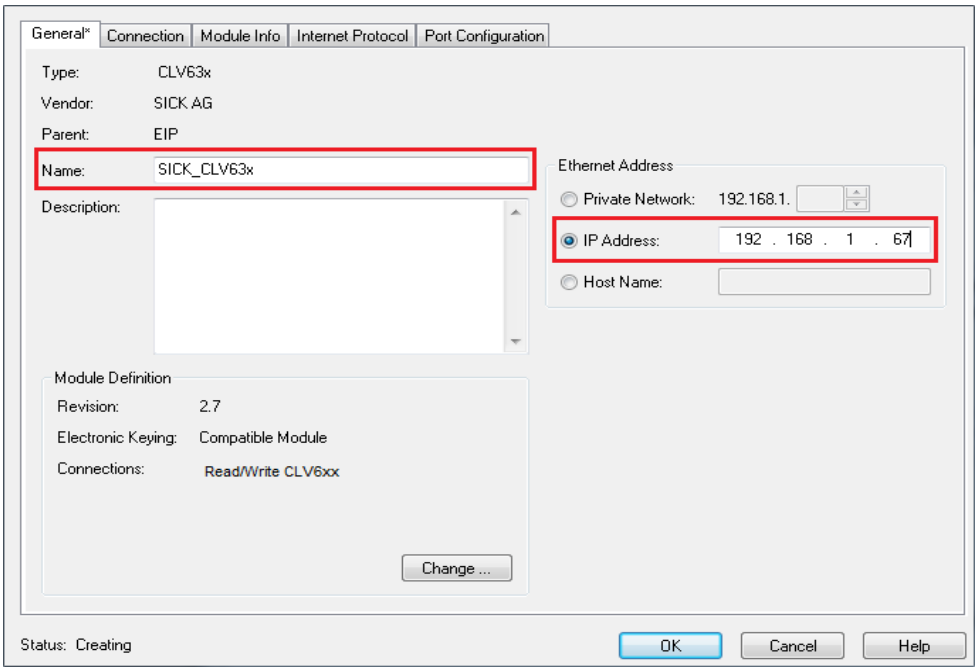


Once the **EDS** file has been registered within the controller's programming software the device will become available when a "**New Module**" is added to the controller's "**I/O Configuration**".

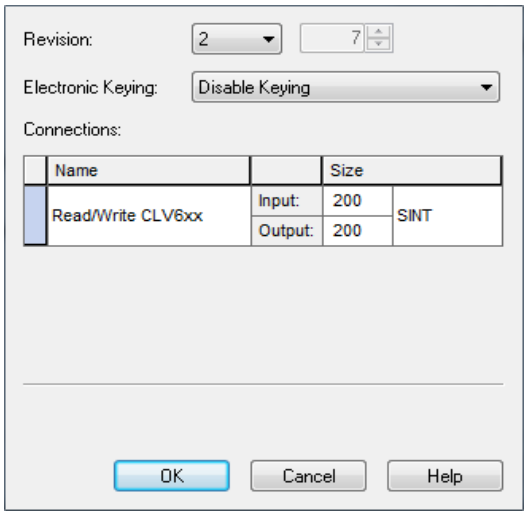
The CLV63x Bar Code scanner would appear as a "**Communications Adapter**" where the vendor would be "**SICK AG**".



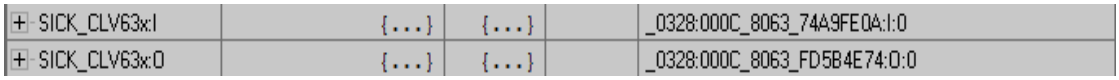
Upon selecting the CLV63x Bar Code Scanner from the "**Catalog**" list a '**New Module**' window will appear where the device properties can be configured.



The data size generated by the **EDS** file can be viewed under '**Module Definition**' when the "**Change…**" button is selected.



Where the data size for the Input and Output can be modified (8 Bytes to 500 Bytes with a default size of 200 Bytes). This will just generate two tags within the programmable controller's "**Controller Tags**".



Along with only generating two tags within the programmable controller's "**Controller Tags**" a "**Connection Faulted**" status is also provided within the Input Tag Database.

SICK_CLV63x:I	{...}	{...}	
SICK_CLV63x:I.ConnectionFaulted	0		Decimal
SICK_CLV63x:I.Data	{...}	{...}	Decimal
SICK_CLV63x:I.Data[0]	73		Decimal
SICK_CLV63x:I.Data[1]	-124		Decimal
SICK_CLV63x:I.Data[2]	4		Decimal
SICK_CLV63x:I.Data[3]	0		Decimal
SICK_CLV63x:I.Data[4]	0		Decimal
SICK_CLV63x:I.Data[5]	0		Decimal
SICK_CLV63x:I.Data[6]	0		Decimal
SICK_CLV63x:I.Data[7]	0		Decimal
SICK_CLV63x:I.Data[8]	0		Decimal
SICK_CLV63x:I.Data[9]	0		Decimal
SICK_CLV63x:I.Data[10]	0		Decimal

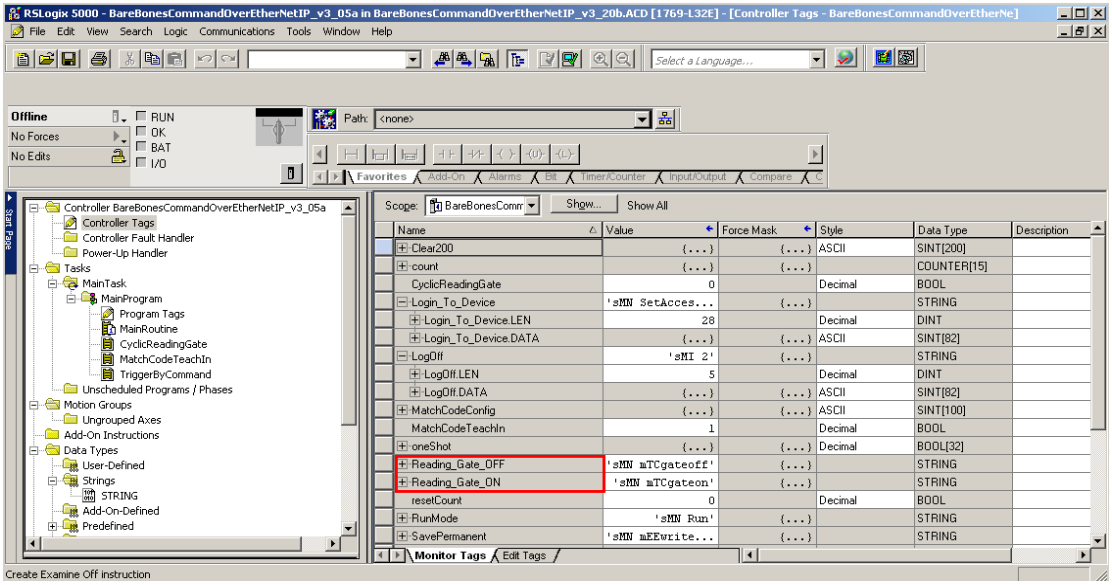
C. Triggering

The actual triggering of the scanner can be done in a few different ways. For the following example a CLV6xx Bar Code Scanner will be triggered using '**Command**' over EtherNet/IP. This choice is made here mostly for the purpose of illustrating, along with the programmable controller code, the possibility of sending commands to the scanner from the controller over EtherNet/IP.

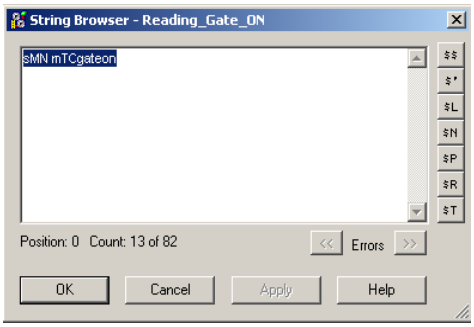
Under normal conditions, it would be recommended to use a hardwired input for the trigger. Another alternative would be to use the Fieldbus Input bit (described below) if it is desirable to trigger the scanner over EtherNet/IP.

Two User Defined Tags (UDTs) are created in the programmable controller. These UDTs have the data type '**String**'. This data type primarily contains ASCII data. In this example the two String UDTs are named "**Reading_Gate_ON**" and "**Reading_Gate_OFF**".

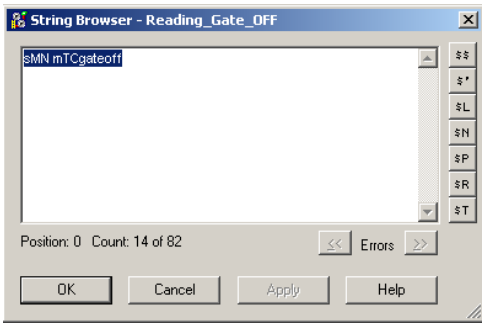
NOTE: The device's 'Start' trigger within its 'Object Trigger Control' parameter must be set to "Command" using SOPAS in order to trigger by command.



The Command string used to Start and Stop the reading gate are:

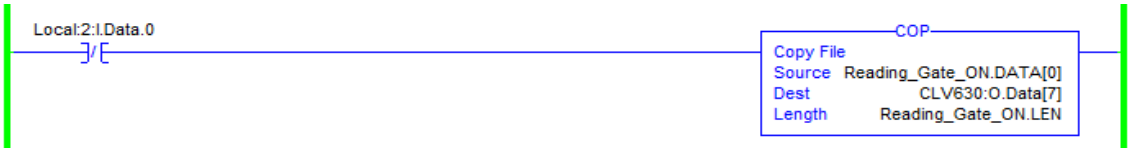


"sMN mTCgateon"

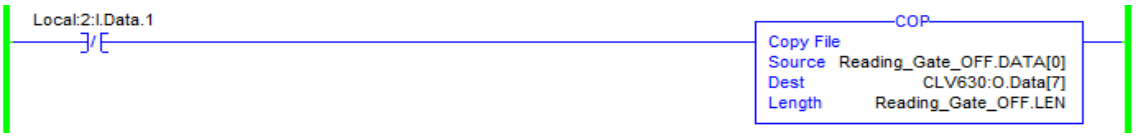


"sMN mTCgateoff"

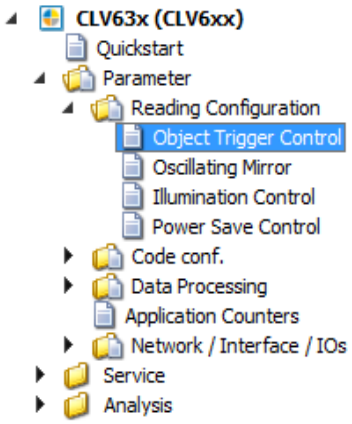
To trigger the CLV6xx Bar Code Scanner by Command the a copy instruction would be used to copy the content in the "**Reading_Gate_ON**" UDT to the Output Tag Database of the CLV bar code reader.



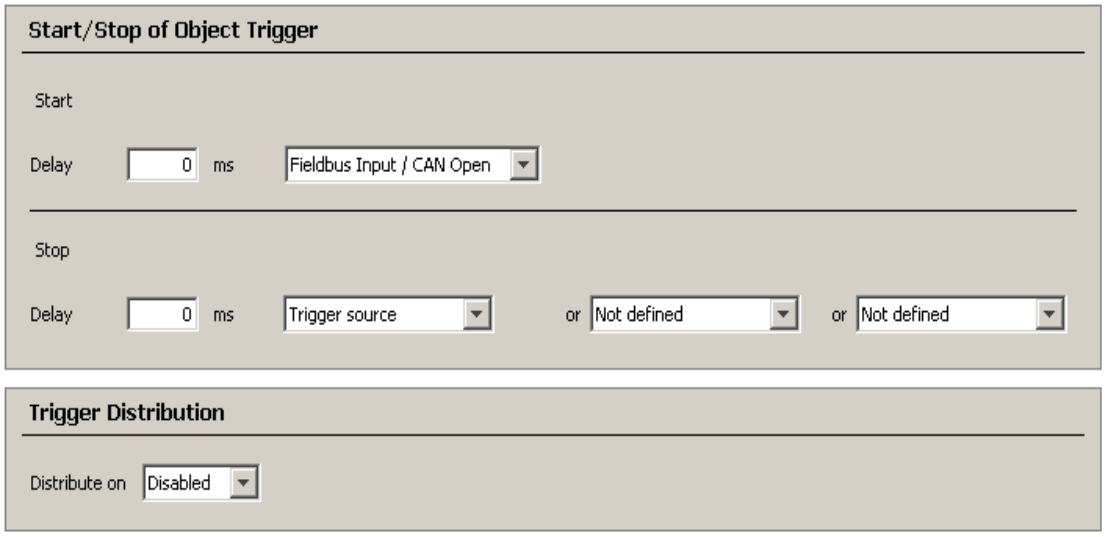
To close the reading gate a copy instruction would be used to copy the content of "**Reading_Gate_OFF**" UDT into the Output Tag Database of the CLV bar code reader.



The CLV6xx Bar Code Scanners can also be triggered to scan by accepting a signal over EtherNet/IP. This must be activated in the CLV6xx Bar Code Scanner.



Where a Trigger can be selected to 'Start' the scanner and a Trigger can be selected to 'Stop' the scanner.



By default the “Start” parameter is set to “Sensor1”. Where a sensor is wired into the connection module at the terminal of the same designation. However, it might be convenient (or necessary) to trigger the scanner utilizing an output data bit in the EIP output data assembly. (refer to **‘Input and Output Data Assemblies’** Part **B** - **Output Data Format**). If an application requires the SICK AutoID device to be triggered in this fashion, select ‘**Fieldbus Input / CAN Open**’ as shown above. This enables the ‘Trigger’ bit in the output data format (see detailed description in ‘**Input and Output Data Assemblies**’ Part **B** - **Output Data Format)** to activate the reading gate of the SICK AutoID device. So if the bit is set = 1 then the reading gate opens. Once it is reset = 0 the reading gate closes.

D. EtherNet/IP Compatibility

This document details the EtherNet/IP capability of SICK AutoID devices using Implicit Message connection. This message connection between the field devices (CLV6xx, LECTOR6xx, RFH6xx, and RFU6xx) and the EtherNet/IP interface on the programmable controller is based on an update time referred to as the Requested Packet Interval (RPI).

SICK AutoID devices do not support EtherNet/IP Explicit Message connection. If this is required please contact SICK Technical Support at.

http://supportportal.sick.com

Use SOPAS-ET configuration software to setup SICK AutoID devices. **SOPAS-ET** = **S**ICK **O**pen **P**ortal **A**pplication and **S**ystem **E**ngineering **T**ool. It can be downloaded from:

https://www.sick.com/us/en/sopas-engineering-tool-v3/p/p367244

Lastly, online information on Rockwell Automation Programmable Controllers, network interfaces, and software can be found at:

http://www.rockwellautomation.com/support/americas/index_en.html